Knowledge Enrichment by Fusing Representations for Malware Threat Intelligence and Behavior

Aritran Piplai*, Sudip Mittal†, Mahmoud Abdelsalam‡, Maanak Gupta§, Anupam Joshi*, Tim Finin*
*Dept. of Computer Science & Electrical Engineering, University of Maryland, Baltimore County,
Email: {apiplai1, joshi, finin}@umbc.edu
†Dept. of Computer Science, University of North Carolina Wilmington, Email: mittals@uncw.edu
‡Dept. of Computer Science, Manhattan College, Email: mabdelsalam01@manhattan.edu
§Dept. of Computer Science, Tennessee Technological University, Email: mgupta@tntech.edu

Abstract—**Security engineers and researchers use their disparate knowledge and discretion to identify malware present in a system. Sometimes, they may also use previously extracted knowledge and available Cyber Threat Intelligence (CTI) about known attacks to establish a pattern. To aid in this process, they need knowledge about malware behavior mapped to the available CTI. Such mappings enrich our representations and also helps verify the information. In this paper, we describe how we retrieve malware samples and execute them in a local system. The tracked malware behavior is represented in our Cybersecurity Knowledge Graph (CKG), so that a security professional can reason with behavioral information present in the graph and draw parallels with that information. We also merge the behavioral information with knowledge extracted from the text in CTI sources like technical reports and blogs about the same malware to improve the reasoning capabilities of our CKG significantly.**

I. Introduction

Representing cybersecurity knowledge in knowledge graphs allows security researchers to query, retrieve, and reason with malware information integrated from multiple sources. In this paper we focus on Open Source Intelligence (OSINT) and malware behavior data as the two main sources for this knowledge. OSINT is information about cyber-attacks available publicly on the Internet, typically collected from vendor reports, blogs, technical reports, and social media [24], [33]. For additional meaningful information about malware behavior we collect malware samples, detonate them in a controlled environment, and then observe the trends of specific system parameters.

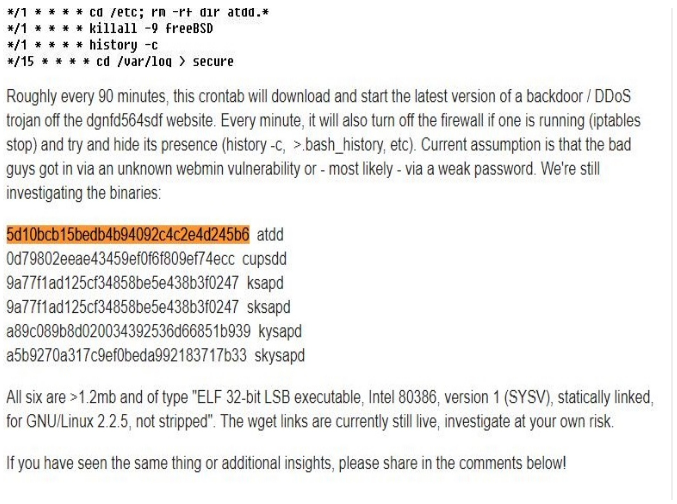
While OSINT has been represented in knowledge graphs before [28], it has been limited to descriptions of analysis performed by security researchers. Fusing this with behavior data adds critical information and forms a better model, as information from the two sources complement each other. Knowledge gathered from OSINT sources may be informal, often lacks in-depth analysis, and is vulnerable to poisoning attacks [14]. Conversely, malware behavior data can be too detailed or overly specific to a given system configuration.

We describe an approach to constructing a cybersecurity knowledge graph (CKG) by integrating the information and correcting some of these problems. Figure 1a shows an example of malware CTI that gives an overview of the malware functionality but lacks incisive analysis. Our method adds more granular information to the CTI information in the form of malware behavior data. There are two key factors for which we are motivated to use our proposed method.

Enriched CKG: A CKG that has been built from the information extracted from OSINT text is limited to the information provided by the author of the text. However, as seen in Figure 1a, an author sometimes focuses on a particular aspect of the malware. For example, the writer of this text does not say how she suspects that the firewall has been compromised nor how she determined that the file was being downloaded every 90 minutes. However, she does infer that the malware has been using a 'weak password' vulnerability, which is something that may not be easily inferred by simply observing the system parameters of the behavior data. If we merge the information that has been stated online about the malware with the behavior data, we can get an *enriched* CKG. The new CKG, shown in Fig 2, integrates information that includes human intuition, along with details about the system parameters which when observed can raise flags about the presence of a malware.

Verification of Information: If we only consider information extracted from open-source text, our CKG will have to rely on the conclusions derived by the authors of the text. For text written by a trusted security organization, we can say with some confidence that the information is likely to be correct. However, very often, open source text describing a malware is written by someone whose identity cannot be verified, which raises a question on the quality of information present in the knowledge graph. Since our knowledge graph's underlying schemas are based on ontologies, constraints, and rules defined in OWL [11], contradictory information can be detected. If an author claims that the network activity will increase after the malware infects a system, and our behavior data says that there has been no increase in the network activity, we can infer that there is a *reasoning error*. This will lead us to conclude that the information stated by the author may not be correct, thereby improving the reliability of our CKG.

The rest of the paper is organized as follows - Section II, describes some related work. We describe our methodology for intelligence extraction and behavior collection in Section III. Knowledge graph reason has been discussed in Section IV. We conclude in Section V.



```
*/1 * * * * cd /etc; rm -rf dir atdd.*
*/1 * * * * killall -9 freeBSD
*/1 * * * * history -c
*/15 * * * * cd /var/log > secure
```

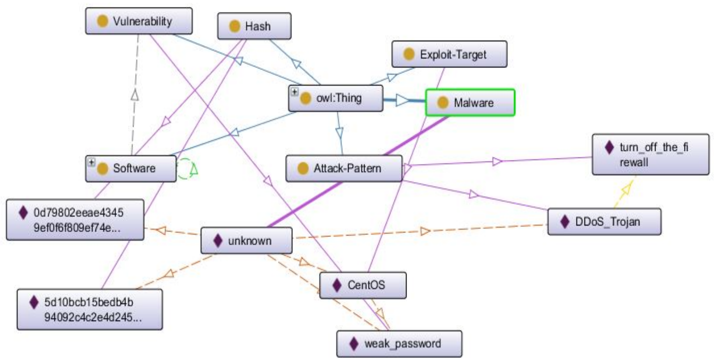
Roughly every 90 minutes, this crontab will download and start the latest version of a backdoor / DDoS trojan off the dgnfd564sdf website. Every minute, it will also turn off the firewall if one is running (iptables stop) and try and hide its presence (history -c, >.bash_history, etc). Current assumption is that the bad guys got in via an unknown webmin vulnerability or - most likely - via a weak password. We're still investigating the binaries:

5d10bcb15bedb4b94092c4c2e4d245b6 atdd
0d79802eeae43459ef0f6f809ef74ecc cupsdd
9a77f1ad125cf34858be5e438b3f0247 ksapd
9a77f1ad125cf34858be5e438b3f0247 sksapd
a89c089b8d020034392536d66851b939 kysapd
a5b9270a317c9ef0beda992183717b33 skysapd

All six are >1.2mb and of type "ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.2.5, not stripped". The wget links are currently still live, investigate at your own risk.

If you have seen the same thing or additional insights, please share in the comments below!

(a)



(b)

Fig. 1: 1a) An excerpt of a CTI describing a malware. 1b) The knowledge graph representation of the CTI.

II. Related Work

We briefly discuss some related research on CTI, malware behavior data, and cybersecurity knowledge graphs.

A. Malware Behavior Data

Behavioral analysis is a popular way to detect malware which primarily focuses on detecting malicious behaviors that impact a specific set of features. Malware detection approaches that use behavioral analysis can be categorized into five categories based on these targeted features:

- **Network features** [6], [22]: such approaches rely on using network features to detect malicious traffic patterns;
- **System calls** [3], [7], [29], [38]: approaches relying on system calls to detect particular sequences of system calls that generally used by malware;
- **Memory features** [25], [39]: approaches using features like malicious memory access patterns to detect malware;
- **Hardware performance counters** [5], [8]: approaches with performance counters (e.g., cache hit/miss) that detect malware;
- **Performance metrics** [2], [18], [36], [37]: approaches using system metrics like CPU or memory utilization to model normal and malicious application behavior.

We use performance metrics data and focus on features that can easily be obtained by a light-weight, on-host agent.

B. CTI and CKG

Cybersecurity Knowledge Graphs have been widely used to represent Cyber Threat Intelligence. CKG stores CTI as an entity-relationship set. It is an aggregation of semantic triples that helps in understanding how different cyber entities or events are related. A semantic triple is of the form 'entity-A' - has relationship 'R' - 'entity-B'. This representation helps with giving users the capability to query the system to retrieve the information they desire about a cyber-event. The classes and possible relationships between their instances are dictated by a pre-defined schema. This schema also helps in reasoning, as it can help define specific constraints and axioms that may exist between the classes. Knowledge graphs for cybersecurity have been developed before, as can be seen in Unified Cybersecurity Ontology (UCO) [27]. Open source intelligence has been used to build CKGs and other agents to aid cyber analyst [19]–[21], [23], [32]. CKGs have also been used to map out the activity of different malware to see how they compare. Liu et al. [17] have used malware features in a knowledge graph to compare malware instances. Graph based methods have also been post processed by machine learning algorithms, as demonstrated by another approach [4], [12], [13]. Some behavioral aspects have also been incorporated in CKGs, where the authors used system call information it in a graph [26].

III. Methodology

Our method fuses CTI information, extracted from text descriptions of a malware, with malware behavior data that has been captured after detonating a sample of the malware in a virtual machine (VM) which is connected to the Internet in a cloud environment. The malware text descriptions are available online, in the form of blogs, community posts and threads about malware, and malware analysis reports [30], [30], [35]. Figure 3, gives the overall architecture of our system. The sample collection mechanism is described in Section III-A. We collect the behavior data features (Section III-B), extract the feature trends of the data (Section III-D2), and ingest them into a behavioral knowledge graph. Simultaneously, we look for open source text describing malware samples which matches the MD5 hash of the malware sample being detonated. We extract CTI from text description in the form of a CTI schema described in Section III-C. We fuse semantic triples of behavior data and CTI into an enriched 'Fused CKG' (Section III-D). Next, we describe individual components in detail.

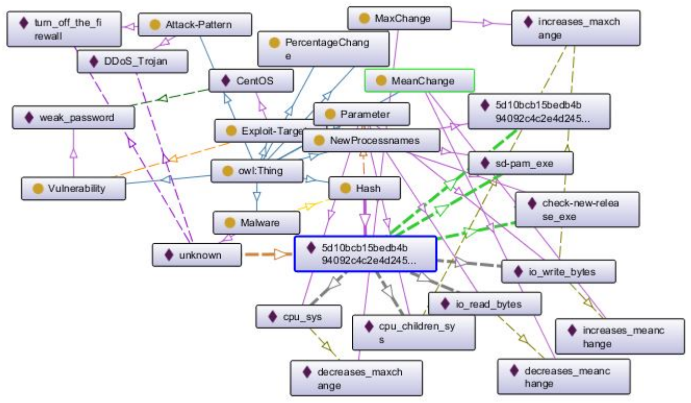


Fig. 2: Fused KG. The bold gray and green lines show the new relations added by fusing the behavior data with the original OSINT data in Fig 1b. The entities on the left side of the graph like Exploit-Target:'CentOS', Attack-Pattern:'turn off the firewall', etc. were extracted from the CTI mentioned above. The behavior data adds key information like processes 'sd-pam.exe', '5d10...exe' getting created, and the trends of certain I/O and CPU parameters.

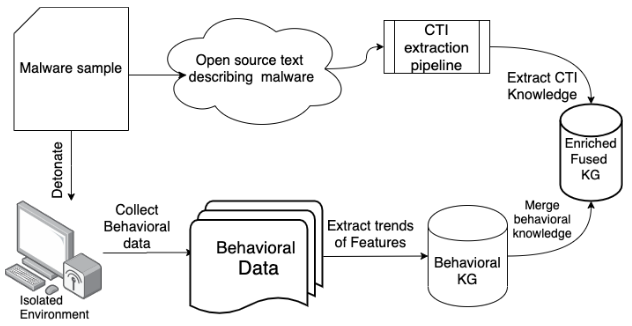


Fig. 3: Architecture diagram describing our system. Malware behavioral data samples were collected by injecting the malware into VMs working in a real environment connected to the Internet.

A. *Malware Behavioral Data Samples Collection*

In this subsection, we describe (1) the malware dataset used in our experiments, (2) the deployed testbed, and (3) the developed data collection module.

1) Malware Executables Dataset: Malware executables are downloaded from VirusTotal[1]. We developed a script to select only working malware that considered any executable that ran successfully to be a working malware instance. However, we cannot say with certainty if malware actually did malicious actions during a run or just remained idle. After filtering, the dataset contains 114 random working malware executables representing eleven different types: Backdoor, Constructor, DDoS/DoS, Exploit, Trojan, Virus, Worm, HackTool, Net-Worm, Trojan-DDoS, and VirTool.

2) Experiment Setup: An Openstack[2] based cloud testbed was used in this experiment. The cloud environment consisted a controller, a network, and three compute nodes comprising 48 cores and 160 GB of memory. Each experiment was done by running a single VM for a total of 30 minutes. An Ubuntu 16.04 image running a web application was used as the victim VM. The first fifteen minutes is referred to as the *benign phase* where the VM was running without malicious activity. The next fifteen is referred to as the *malicious phase* where a malware was injected and executed into the VM. To simulate a VM workload, a traffic generator was developed and used to simulate multiple clients communicating with the VM by sending random POST and GET requests. This workload is following a self-similar [15] ON/OFF Pareto distribution.

3) Data Collection: A module was integrated into the Ubuntu image to collect various VM's performance metrics. This data was sent in the form of JSON objects to an outside storage machine. The data agent module collected metrics in ten seconds intervals where a total of 180 samples of features were recorded during a single experiment. A single experiment was conducted for each malware executable, thus producing a total of 114 experiments. After each experiment, the VM is deleted to avoid contamination of clean data in the subsequent experiment runs. During all our experiments, we allowed Internet connection and disabled all firewalls to ensure the malware malicious activity. This is due to the fact that most sophisticated malware instances suppress their malicious activity if they cannot reach their command and control server [16].

B. *Behavior Data*

To carry out their malicious activities, malware relies on system resources that can range from negligible to substantial resources utilization. For example, ransomware uses a considerable amount of CPU, disk and memory to encrypt the targeted files, while command and control malware sends and receives a relatively small number of network packets. Nevertheless, all malware uses some system resources in one way or another. We rely on performance metrics as an important sensor to characterize malware behaviors. This data depends on volumetric system information (e.g. CPU utilization, number of packets/bytes sent and received, and memory utilization) which gives a generic overview of the impact of malware on systems resources.

Table I gives our behavioral data metrics, which form the time series trends of the operating system's processes. The selected metrics are used to show the effectiveness of our approach; however, additional metrics are available in practice which can give more in-depth insight into malware behavior. We grouped the selected metrics into eight categories: (1) *status*: indicating the status of processes as sleep, active, etc., (2) *CPU information*: giving the detailed CPU utilization of processes (e.g., CPU utilization spent in kernel and user space), (3) *Context switches*: identifying the context switches

[1]VirusTotal. https://www.virustotal.com/

[2]Openstack. https://www.openstack.org/

TABLE I: Virtual machines performance metrics [1].

Metric Category	Description
Status	Process status
CPU information	CPU usage percent, CPU times in user space, CPU times in system/kernel space, CPU times of children processes in user space, CPU times of children processes in system space.
Context switches	Number of context switches voluntary, Number of context switches involuntary
IO counters	Number of read requests, Number of write requests, Number of read bytes, Number of written bytes, Number of read chars, Number of written chars
Memory information	Amount of memory swapped out to disk, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Number of dirty pages, Amount of physical memory, text resident set (TRS), Memory used by shared libraries, memory that with other processes
Threads	Number of used threads
File descriptors	Number of opened file descriptors
Network information	Number of received bytes, Number of sent bytes

TABLE II: Modified UCO Classes and Relationships.

Type	List
Classes	Software, Exploit-Target, Malware, Indicator, Vulnerability, Course-of-Action, Tool, Attack-Pattern, Campaign, Filename, Hash, IP Addresses
Relationships	attributedTo, indicates, hasProduct, hasHash, mitigates, hasVulnerability, uses

done voluntarily and involuntarily by processes, (4) *IO counters*: indicating disk utilization information (e.g., the number of bytes written and read, etc.), (5) *Memory information*: showing the processes memory utilization metrics (e.g., amount of memory swapped out, memory used by shared libraries, etc.), (6) *Threads*: giving the number of threads created in processes, (7) *File descriptors*: representing the number of file descriptors used by processes, and (8) *Network information*: indicating processes network utilization metrics such as the number of bytes sent and received.

C. CTI Extraction Pipeline: Updating UCO 2.0

Unified Cybersecurity Ontology (UCO) [27], based on Structured Threat Information Expression (STIX 1.0) [10], describes a schema to represent CTI. UCO 2.0 is an updated version of the ontology which is based on the updated version of STIX (version 1.2) [9]. The classes and relationships have been modified, in accordance with work by Pingle et al. [27], which best represents CTI extracted from cybersecurity technical reports. The modified **classes** and **relationships** are given in Table II and briefly described below.

A *Software* is a piece of published program that can be used or targeted by a malware, like Microsoft Office. An *Exploit-Target* usually relates to the site of the attack and is typically an operating system like Windows or Ubuntu. *Indicators* are known cues about a malicious attack. A *vulnerability* can be a bug or a general weakness that can be exploited by malware. *Course-of-Action* denotes a sequence of steps that a user can or must perform to mitigate an attack. *Filenames* and *IPAddresses* are known examples that a malware uses during the attack. An *Attack-Pattern* is a sequence of activities that a malware performs during the attack.

The *Hash* is a key class for our purpose as it denotes one of the known MD5 hash values associated with the malware. We use it to merge the CTI information with the behavior data of the detonated malware. The *attributedTo* relationship is about a campaign that is related to a Malware, Tool, or Vulnerability. An *Indicates* relationship holds between an Indicator and a Malware, or a Tool. *hasProduct* is a relationship between two Software instances. *Course-of-Action* mitigates a Malware or a Tool. *hasHash* is a relationship between a Malware and a Hash. The relationship *hasVulnerability* exists between a Software, or an Exploit-Target, with a Vulnerability. A Malware or a Tool uses an *Attack-Pattern* or another *Tool*. These ontology components represents the schema for our Cybersecurity Knowledge Graph (CKG) which captures CTI from text data.

D. Representation of data in a Fused CKG

We have an established pipeline that extracts data from open-source technical reports or blogs. The pipeline extracts the cybersecurity-relevant entities and the relationships between them from the text and assigns them to classes of the CKG schema based on STIX. The behavioral data is indexed based on MD5 hashes of the malware detonated in VM. We search for reports or blogs available on the Internet that mention the MD5 hashes and run those reports through the pipeline. This results in an entity-relation set about knowledge extracted from text reports scraped from the internet.

1) CTI extraction from cybersecurity text: The pipeline takes cybersecurity text, which can be in the form of blogs or reports published by verified or unverified sources, as input and extracts cyber entities mentioned in the report. The cyber entity extraction is based on a Named Entity Recognizer (NER) trained on a cybersecurity corpus. We then pass the extracted cyber entities to a machine learning model which establishes a relationship existing between each pair of entities [19], [20], [27]. This results in an entity-relationship set, which tells us facts about the malware mentioned in the text are inter-related. This representation of CTI as a knowledge graph helps us to to integrate and reason over the extracted knowledge. It also provides interfaces to run queries that can be used to retrieve the extracted information.

2) Behavioral data extraction: As mentioned in Section III-A, we observe the parameters in Table I every ten seconds. After a time interval, we detonate a malware sample and keep on observing the same parameters. However, this results in a large volume of data for one malware that is not suitable to be asserted into a knowledge graph. A security analyst interested in the behavior of a particular malware may find it more useful to retrieve information about the pattern certain system parameters exhibit when a malware is running in the system. Moreover, for some system parameters, like memory, context switches are specific to the system.

The exact value of these parameters might not be useful but their *trend* may be very useful from a practitioner's perspective. We extract the parameter trends from the malware behavior data and ingest them into the "Behavior Knowledge Graph". We separate the behavioral data feed for each malware into two sub-groups. The first has data from the time observation starts till the timestamp just before the malware gets detonated and the second includes data from the timestamp when the malware gets detonated, till the end of observation. Once we have these two sub-groups, we can calculate how the time-series trends differ.

Difference in the value sets between one group and another: We use this to identify the difference between features like 'pid' and 'name' (process names). For process id and process names we simply check the count and the new process names respectively, that are being created after the malware is active in the system.

Difference between the average, min, max values: We use this for numerical features of our behavioral data. We take the difference between the average of the features, before and after the malware detonation. This is particularly useful for fields related to memory and 'context switches'. If the average value of these features jumps by a significant amount, we record that in our Behavioral KG. We also calculate the difference between the min and max of certain features. This helps us to record unnatural spikes for certain features of the data. This is particularly useful for features like 'CPU usage'.

Correlation Coefficient: This metric is used to compare the time series trends of selected features before and after the malware detonation. If the correlation coefficient for one particular feature is high positive, then we can say that the feature remained unaffected by the malware detonation. A strong correlation between them would mean that the same trend follows even after malware detonation. However, if the correlation coefficient is negative or very close to zero, we can say that the time series trend has changed significantly after the detonation of the malware. We check this value before we assert the previous values to the Behavioral KG.

We form a super-set of the classes and relationships present in the schema of our Behavioral KG, and the classes of relationships of UCO 2.0. The merged set of classes and relationships has the ability to capture and represent data coming from CTI as well as the behavioral parameters. We search for the MD5 hash, on which the behavior data is indexed, among open source text descriptors about malware. If there is a match, we sent the text through the CTI extraction pipeline. We then assert the entire set of entities and relationships (both behavior and CTI) to the fused CKG.

IV. Knowledge Graph Reasoning

The behavioral knowledge when represented in a knowledge graph, presents us with query and reasoning capabilities. We run SPARQL [31] queries to extract the exact information we desire. For example, if we simply want to query from the information asserted in the behavioral knowledge graph, we can run the following query. This simply translates to 'which parameters have their maximum values changed for the malware which has the hash: '5d10bcb15bedb4b94092c4c2e4d245b6'. A point to be noted here is that the values returned have their maximum values changed by a significant amount. For our experiments, we chose this threshold as 30%.

```
SELECT ?x WHERE {
  BKG:5d10bcb15bedb4b94092c4c2e4d245b6
     BKG:hasParameter ?x.
  ?x BKG:parameterchange
     BKG:increases_maxchange.}
```

This query results in two values:

```
cpu_children_sys,io_write_bytes
```

The reasoning capabilities are enhanced when the behavioral data is fused with the CTI extracted from text descriptions of the malware with the same MD5 hash. The following query asks a more complex question: 'Return all attack patterns that a malware uses and the parameters which show high mean change and the parameters should be associated with an MD5 hash which the malware uses'.

```
SELECT  DISTINCT ?z ?y WHERE {
  ?x a FusedKG:Malware.
     FusedKG:hasHash ?z.
     FusedKG:uses ?y.
  ?y a FusedKG:Attack-Pattern.
  ?z FusedKG:parameterchange;
     FusedKG:increases_meanchange.}
```

which returns the following three values:

```
DDoS_Trojan,turn_off_firewall,io_write_bytes
```

We observe that the fused knowledge graph answers the question based on the information extracted from both CTI and the behavioral data. The attack patterns are extracted from a blog written by the user about this malware which has the same MD5 hash as the malware detonated for the observation of the behavior parameters. This reflects how the CKG gets more enriched after fusion.

V. Conclusion and Future Work

We successfully demonstrated how fusing CTI information with malware behavior data enriches our CKG. Our fused CKG captures information about analyses performed by users and also the trends of various system parameters of the behavior data. Removal of contradictory statements is enforced by the reasoning capabilities of proposed CKG, and thus improves the veracity of the data present in the CKG. The fused CKG helps in aggregating data from OSINT and malware behavior, which is detonated in our local systems, and thus aids in the process of discovering more information than what was present from the individual sources. We believe that our enriched knowledge gathered from OSINT and malware behavior addresses some of the key shortcomings of the previous research conducted in this area.

We plan to improve our system in several ways. We will enhance our system for ingesting CTI data from more feeds and automatically update the CKG on regular basis. We will revise our approach to encoding provenance to use new features of RDF*, which will facilitate exporting our CKGs to systems based on property graphs, such as Neo4j. We will also incorporate our work adding similarity metrics based on graph embeddings and tensor decomposition and explore how these can be used.

Acknowledgement

This work was supported by National Science Foundation awards 2025685, 2025682, and 2025686, a grant from U.S. Department of Defence, and a gift from IBM Research.

References

[1] Mahmoud Abdelsalam, Ram Krishnan, Yufei Huang, and Ravi Sandhu. Malware detection in cloud infrastructures using convolutional neural networks. In *11th Int. Conf. on Cloud Computing*. IEEE, 2018.

[2] Mahmoud Abdelsalam, Ram Krishnan, and Ravi Sandhu. Online malware detection in cloud auto-scaling systems using shallow convolutional neural networks. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 381–397. Springer, Cham, 2019.

[3] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, Moutaz Alazab, et al. Zero-day malware detection based on supervised learning algorithms of API call signatures. In *Ninth Australasian Data Mining Conference*. Australian Computer Society, 2011.

[4] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(1):247–258, 2011.

[5] Mohammad Bagher Bahador, Mahdi Abadi, and Asghar Tajoddin. Hpc-malhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *4th Int. Conf. on Computer and Knowledge Engineering*, pages 703–708. IEEE, 2014.

[6] Dimitra Chamou, Petros Toupas, Eleni Ketzaki, Stavros Papadopoulos, Konstantinos M Giannoutakis, Anastasios Drosou, and Dimitrios Tzovaras. Intrusion detection system based on network traffic using deep neural networks. In *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2019.

[7] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2013.

[8] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[9] Oasis group. Stix 2.0 documentation. https://oasis-open.github.io/cti-documentation/stix/examples.html, May 2013.

[10] Oasis group. Stix 1.0 documentation. https://stixproject.github.io/-documentation/, May 2018.

[11] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, and Sebastian Rudolph. OWL 2 web ontology language primer, 2012.

[12] Karuna P Joshi, Aditi Gupta, Sudip Mittal, Claudia Pearce, and Tim Finin. Alda: Cognitive assistant for legal document analytics. In *AAAI Fall Symposium on Cognitive Assistance in Government and Public Sector Applications*. AAAI Press, 2016.

[13] Maithilee Joshi, Sudip Mittal, Karuna P Joshi, and Tim Finin. Semantically rich, oblivious access control using abac for secure cloud storage. In *Int. conf. on edge computing*, pages 142–149. IEEE, 2017.

[14] Nitika Khurana, Sudip Mittal, Aritran Piplai, and Anupam Joshi. Preventing poisoning attacks on AI based threat intelligence systems. In *Int. Workshop on Machine Learning for Signal Processing*. IEEE, 2019.

[15] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking*, 2(1):1–15, 1994.

[16] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *Int. Workshop on Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[17] Jing Liu, Yuan Wang, and Yongjun Wang. The similarity analysis of malicious software. In *Int. Conf. on Data Science in Cyberspace*. IEEE, 2016.

[18] Andrew McDole, Mahmoud Abdelsalam, Maanak Gupta, and Sudip Mittal. Analyzing cnn based behavioural malware detection techniques on cloud iaas. In *Int. Conf. on Cloud Computing*. IEEE, 2020.

[19] Sudip Mittal, Prajit Das, Varish Mulwad, Anupam Joshi, and Tim Finin. Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE Press, 2016.

[20] Sudip Mittal, Anupam Joshi, and Tim Finin. Thinking, fast and slow: Combining vector spaces and knowledge graphs. *arXiv preprint arXiv:1708.03310*, 2017.

[21] Sudip Mittal, Anupam Joshi, and Tim Finin. Cyber-all-intel: An ai for security related threat intelligence. *preprint arXiv:1905.02895*, 2019.

[22] SH Mousavi, Mohammad Khansari, and R Rahmani. A fully scalable big data framework for botnet detection based on network traffic analysis. *Information Sciences*, 512:629–640, 2020.

[23] Lorenzo Neil, Sudip Mittal, and Anupam Joshi. Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports. In *Int. Conf. on Intelligence and Security Informatics*. IEEE, 2018.

[24] Federico Neri and Paolo Geraci. Mining textual data to boost information access in OSINT. In *Int. Conf. on Information Visualisation*. IEEE, 2009.

[25] Meltem Ozsoy, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Malware-aware processors: A framework for efficient online malware detection. In *21st International Symposium on High Performance Computer Architecture*, pages 651–661. IEEE, 2015.

[26] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. Fast malware classification by automated behavioral graph matching. In *6th Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2010.

[27] Aditya Pingle, Aritran Piplai, Sudip Mittal, Anupam Joshi, James Holt, and Richard Zak. Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. In *Int. Conf. on Advances in Social Networks Analysis and Mining*. IEEE, 2019.

[28] Aritran Piplai, Sudip Mittal, Anupam Joshi, Tim Finin, James Holt, and Richard Zak. Creating cybersecurity knowledge graphs from malware after action reports. *UMBC Faculty Collection*, 2019.

[29] Radu Pirscoveanu, Steven Hansen, Thor Larsen, Matija Stevanovic, Jens Myrup Pedersen, and Alexandre Czech. Analysis of malware behavior: Type classification using machine learning. In *Int. conf. on cyber situational awareness,data analytics and assessment*. IEEE, 2015.

[30] Line Pouchard, Jonathan Dobson, and Joseph Trien. A framework for the systematic collection of open source intelligence. In *AAAI Spring Symposium on Technosocial Predictive Analytics*, 2009.

[31] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language. http://www.w3.org/TR/rdf-sparql-query/.

[32] Priyanka Ranade, Sudip Mittal, Anupam Joshi, and Karuna Joshi. Using deep neural networks to translate multi-lingual threat intelligence. In *Int. Conf. on Intelligence and Security Informatics*. IEEE, 2018.

[33] Robert David Steele. Open source intelligence: What is it? why is it important to the military. *American Intelligence Journal*, 17(1), 1996.

[34] Zareen Syed, Ankur Padia, M. Lisa Mathews, Tim Finin, and Anupam Joshi. UCO: A unified cybersecurity ontology. In *AAAI Workshop on Artificial Intelligence for Cyber Security*. AAAI Press, February 2016.

[35] Symantec. 2016 internet security threat report, April 2016.

[36] Shun Tobiyama, Yukiko Yamaguchi, Hajime Shimada, Tomonori Ikuse, and Takeshi Yagi. Malware detection with deep neural network using process behavior. In *40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 577–582. IEEE, 2016.

[37] M. Watson, N. Shirazi, A. Marnerides, A. Mauthe, and D. Hutchison. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing*, 13(2):192–205, 2015.

[38] Wenqi Xie, Shengwei Xu, Shihong Zou, and Jinwen Xi. A system-call behavior language system for malware detection using a sensitivity-based LSTM model. In *3rd International Conference on Computer Science and Software Engineering*, pages 112–118. ACM, 2020.

[39] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. Malware detection using machine learning based analysis of virtual memory access patterns. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 169–174. IEEE, 2017.